Extendiendo los Horizontes de la Computación Distribuida

Javier Echaiz* Jorge R. Ardenghi

Laboratorio de Investigación en Sistemas Distribuidos (LISiDi)
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur – Bahía Blanca, Argentina
T.E.: +54 291–4595135 Fax: +54 291–4595136
{je,jra}@cs.uns.edu.ar

Resumen

Un sistema distribuido está compuesto de nodos, posiblemente heterogéneos, conectados mediante una red. Un sistema de esta clase puede utilizarse efectivamente sólo si el software es capaz de presentar al usuario el concepto de *single system image* (SSI) sobre el sistema físicamente distribuido. De esta forma todos los recursos de un nodo deberían poder accederse transparentemente desde cualquier otro nodo.
En esta línea de investigación se plantea la problemática de un protocolo de migración de procesos como componente fundamental de un sistema distribuido y su extensión al nuevo paradigma de computación: *grid computing*.

Palabras Clave: *grid computing*, *clustering*, carga compartida, balance de carga, migración de procesos, sistemas distribuidos.

1 Introducción

El desarrollo de microprocesadores de elevado poder de cómputo y bajo costo y las redes de alta velocidad propiciaron en las dos últimas décadas un cambio en el paradigma de la computación. Los grandes *mainframes* de ayer fueron reemplazados por clusters de pequeñas pero potentes computadoras conectadas mediante redes de alta velocidad. El usuario, en lugar de trabajar en una terminal boba conectada por un cable serie al mainframe, ahora tiene en su escritorio una poderosa computadora unida con pares mediante una red. Potencialmente todos los recursos que físicamente se encuentran en cualquier computadora están disponibles a cualquier usuario del sistema. Sin embargo este potencial no puede aprovecharse completamente a menos que los usuarios puedan acceder transparentemente a estos recursos. La transparencia, en este contexto, significa que los usuarios deben poder acceder a cualquier recurso sin preocuparse (y de hecho, sin siquiera saber) su ubicación física. Este es el objetivo de un sistema operativo distribuido, crear la ilusión en la mente de los usuarios de un sistema de tiempo compartido único en lugar de una colección de máquinas independientes pero conectadas.

Algunos de los sistemas más destacados son Amoeba [MvRT+90], Sprite [DO91], etc. La mayoría de estos sistemas trabajan corriendo una copia del mismo sistema operativo en cada computadora participante, y estas copias a su vez, cooperan para proveer SSI [EA03] a sus usuarios.

Sin embargo, si bien conceptualmente estos sistemas resultan atractivos, no presentaron un crecimiento real en su popularidad. Esto se debe principalmente al hecho de que ya existe

*Becario del Consejo Nacional de Investigaciones Científicas y Técnicas, República Argentina.

una gran cantidad de usuarios y software de base para UNIX; por lo tanto muchos sistemas distribuidos tratan de emular UNIX para que las aplicaciones existentes sigan funcionando con pocos o sin modificaciones, manteniendo un entorno de trabajo familiar a los usuarios.

Por otra parte el software de los sistemas distribuidos actuales debe resolver otro problema. El problema surge debido a que los sistemas distribuidos actuales están compuestos de una gran variedad de hardware (provisto por diferentes fabricantes) y de software de base (sistema operativo). Lograr SSI sobre esta heterogeneidad constituye un gran reto. Los sistemas como Amoeba, etc. no tratan de resolver el problema de la heterogeneidad de sistemas operativos, simplemente asumen que todas las máquinas participantes del cluster corren el mismo S.O. Hoy en día existen soluciones parciales a este problema, e.g. SunNFS [SGK+85] a nivel del sistema de archivos distribuido con vista unificada. Otro recurso comúnmente compartido en los UNIX actuales son las impresoras.

Sin embargo el CPU *no* suele ser compartido transparentemente en los sistemas UNIX. Existen varios estudios que muestran que existe una gran disparidad en la carga de los nodos de un sistema distribuido en cualquier instante de tiempo. Mientras algunas máquinas están sobrecargadas otras se encuentran completamente ociosas. Es entonces deseable que estas máquinas puedan compartir la carga y así los usuarios podrían observar una mejora en la performance del sistema. Si bien existen comandos UNIX a nivel de usuario (e.g. `rsh`) que les permiten ejecutar sus procesos remotamente en una máquina a su elección, claramente estos mecanismos no son transparentes. En un ambiente ideal un usuario debería simplemente ejecutar una tarea y el sistema automáticamente seleccionar su mejor ubicación (i.e., la máquina menos cargada) para ejecutarla. Una forma más estricta de carga compartida es el llamado *balance de carga*, donde el sistema trata de equiparar la carga de todas las máquinas en todo momento. Mientras que con carga compartida el sistema simplemente debe seleccionar la mejor máquina para ejecutar una tarea recientemente disparada y transparentemente transferirla a esa máquina, con balance de carga típicamente se debería migrar una tarea a otra máquina, posiblemente *durante* su ejecución. Estas dos formas de migración de procesos se conocen como *migración no apropiativa* (o ejecución remota) y *apropiativa* respectivamente.

Obviamente la migración no apropiativa es más sencilla de implementar que la apropiativa. Esto se debe a que para implementar migración apropiativa, el sistema debe efectuar un *checkpoint* en el estado del proceso mientras se encuentra en ejecución y transferir dicho estado a la máquina destino. Claramente este problema no tiene una fácil resolución si las máquinas poseen arquitecturas y/o sistemas operativos diferentes, pues el checkpoint debería efectuarse a nivel del código fuente del programa y no a nivel del código ejecutable. Tui [SH96] es el ejemplo paradigmático de un sistema que permite que procesos en ejecución migren a sistemas con arquitecturas diferentes. Sin embargo, en este caso el proceso de migración no es transparente al programa de aplicación, i.e., la aplicación debe colaborar con el software de migración para que funcione correctamente. Por otra parte la tarea de migración es más costosa en términos de tiempo, pues la totalidad del estado (que puede ser de tamaño considerable) debe transferirse a la máquina destino. Hay estudios que muestran que este overhead adicional restringe severamente los beneficios en la performance que pueden obtenerse mediante la migración apropiativa frente a la no apropiativa [ELZ88]. La ejecución remota no incurre en este costo adicional debido a que únicamente las tareas recientemente disparadas son transferidas a otras máquinas, y por lo tanto no hay espacio de direcciones a transferir. Adicionalmente, en este caso, el problema de la heterogeneidad es más simple de resolver.

La carga compartida trae aparejados dos temas relacionados. El primero se relaciona con las políticas de migración. Por ejemplo, cuándo debe un nodo tratar de transferir un proceso a

otro nodo, qué proceso debe migrarse y a cuál máquina. El segundo tema está constituido por los mecanismos de transferencia de procesos, los cuales aseguran que la tarea migrada obtenga un entorno aproximadamente igual al de la máquina que lo originó.

2 Políticas y Mecanismos de Migración

La planificación (*scheduling*) de tareas en un sistema distribuido de carga compartida involucra decidir no solamente cuándo ejecutar un proceso, sino también dónde ejecutarlo. Para ello la planificación se lleva a cabo mediante dos componentes: el *distribuidor* (políticas de distribución) y el *planificador* (políticas de planificación). El distribuidor decide dónde se ejecutará una tarea y el planificador dictamina cuándo una tarea recibe su porción de CPU. Típicamente cada nodo de un sistema distribuido tiene su propio planificador responsable de organizar los procesos en el(los) procesador(es) local(es), mientras que las decisiones de alto nivel de asignar un proceso a un nodo son tomadas por el distribuidor. Aunque existen pequeñas variaciones [Ous82], este esquema es el que surge naturalmente en un sistema distribuido. Esto se debe a dos motivos, el primero es que cada nodo tiene su sistema operativo propio encargado de planificar procesos, el segundo es la modularidad: los diseñadores pueden concentrarse mejor en los temas relativamente complejos de la distribución de la carga sin necesidad de involucrarse con los detalles de planificación. Las políticas de distribución tratan de repartir la carga total del sistema a sus nodos individuales transfiriendo procesos entre los nodos. Las políticas de planificación simplemente eligen un proceso del conjunto de procesos listos a ejecutarse en un nodo de forma tal de maximizar el *throughput*. La política de planificación de procesos de cualquier sistema operativo tradicional puede funcionar como una política de planificación del sistema distribuido. En realidad la planificación es efectuada por el S.O. por si mismo y por lo tanto debemos concentraremos principalmente en la política de distribución.

La semántica de procesos UNIX no es simple de extender a un entorno distribuido debido a dos razones: el diseño contempla únicamente el caso de un nodo único e independiente, y más importante aún existe una fuerte relación entre el subsistema de procesos y otros subsistemas UNIX [Shi97].

3 Extendiendo el Cluster

Un cluster que cumpla con las características antes mencionadas no sólo proveerá una mayor capacidad de cómputo (al aprovechar CPUs ociosos) sino que además será capaz de lograr un mejor balance en la utilización de recursos, suavizando los picos de actividad.

Otras posibilidades interesantes de la migración de procesos en un cluster se relacionan con un mejor aprovechamiento del espacio de almacenamiento, tolerancia a fallas, localidad de acceso a los datos, mejor administración del sistema, y computación móvil [MDP+00].

El sistema descripto en [EA04], provee migración no apropiativa para clusters heterogéneos tipo UNIX. Esta heterogeneidad se refiere no sólo a distintas configuraciones, sino también a diferentes arquitecturas y sistemas operativos. Estas capacidades presentes en nuestro sistema, pueden extenderse a un ambiente de *grid computing*.

Distintos autores [FG01, DK02, LBB02] han tratado de precisar la definición de *grid*. De hecho, el concepto de *grid computing* se encuentra todavía en evolución y la mayoría de los intentos por precisarlo terminan excluyendo implementaciones que muchos considerarían grids. Es por ello que en este trabajo no se persigue precisar este concepto. En este sentido emplearemos

este término para referirnos a un sistema distribuido geográficamente donde cada componente, en lugar de ser un nodo es un cluster (ver Figura 1).

En la Figura 1 se muestra además un posible ejemplo de uso de este sistema. Supongamos que el Cluster A presenta un índice de estado en sobrecarga, a diferencia de los restantes clusters que se encuentran ociosos. En este caso A migrará tareas para balancear los recursos globales, absorbiendo picos de carga en una parte del sistema. Si bien se trata de un ejemplo sencillo es claro que la complejidad aumenta considerablemente al escalar nuestro cluster inicial a un ambiente grid.

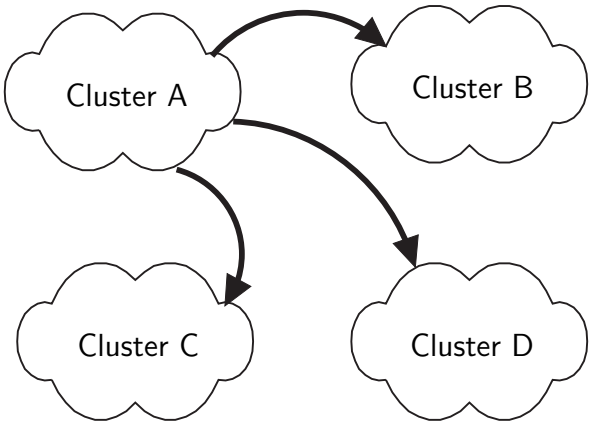

Figura 1: Migran tareas hacia clusters menos cargados para balancear recursos

A primera vista esta nueva tecnología puede verse como una herramienta capaz de tomar cualquier aplicación y ejecutarla 1000 más rápido. Este no es el caso, pues no cualquier aplicación puede paralelizarse. Más aún, adaptar ciertas aplicaciones para obtener un mayor *throughput* en este nuevo ambiente llevaría demasiado esfuerzo. Por otro lado la configuración del sistema podría afectar en gran medida la performance, confiabilidad y seguridad de la infraestructura de cómputo de la organización.

4 Trabajos Futuros

Si bien existe una gran diversidad de opciones al considerar la implementación de un sistema operativo distribuido con capacidad de carga compartida, es nuestra intención implementar un sistema que cumpla con los siguientes objetivos: heterogeneidad a nivel del S.O., migración no apropiativa, cambios mínimos en la semántica UNIX, escalabilidad, eficiencia, y tolerancia a fallas (a nivel del nodo y de la red). Otro objetivo perseguido es mantener independientes los mecanismos de la migración de procesos y las políticas de carga compartida, i.e., dados los mecanismos se debería poder emplear cualquier política de carga compartida. Esta posibilidad permitirá explorar las ventajas/desventajas de diferentes políticas según las necesidades particulares del sistema a considerar. El diseño de este sistema se encuentra actualmente completo.

Una vez verificado y optimizado el protocolo de carga compartida a partir de la implementación[1] se procederá al análisis de las modificaciones y extensiones necesarias para el ambiente grid. Deberán ser consideradas, entre otras, cuestiones relacionadas con la planificación y distribución de la carga, latencias de red, seguridad, y un nuevo modelo de tolerancia a fallas.

[1]En realidad se efectuarán al menos dos implementaciones para verificar la correctitud del protocolo en un ambiente heterogéneo.

Sin duda alguna si bien las tecnologías de *grid computing* se encuentran atravesando sus primeras etapas, muestran una tendencia que, potenciada por el crecimiento de la web, va en claro ascenso.

Bibliografía

[DK02] Ewa Deelman y Carl Kesselman. Grid computing. *Scientific Programming*, 10(2):101–102, 2002.

[DO91] F. Douglis y J. Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice & Experience*, 1991.

[EA03] Javier Echaiz y Jorge Ardenghi. Single System Image: Pilar de los Sistemas de Clustering. *V Workshop de Investigadores en Ciencias de la Computación, WICC 2003*, páginas 210–214, Mayo 2003.

[EA04] Javier Echaiz y Jorge Ardenghi. Carga Compartida en Sistemas Distribuidos Heterogéneos. *V Workshop de Investigadores en Ciencias de la Computación, WICC 2004*, páginas 549–553, Mayo 2004.

[ELZ88] Derek L. Eager, Edward D. Lazowska y John Zahorjan. The Limited Performance Benefits of Migrating Active Processes for Load Sharing. *Conf. on Measurement & Modelling of Comp. Syst., ACM SIGMETRICS*, páginas 63–72, Mayo 1988.

[FG01] Geoffrey Fox y Dennis Gannon. Grid computing: Computational grids. *Computing in Science and Engineering*, 3(4):74–??, Julio/Agosto 2001.

[LBB02] And Domenico Laforenza, Mark Baker y Rajkumar Buyya. Grids and grid technologies for wide-area distributed computing, Julio 31 2002.

[MDP+00] Dejan S. Milojíčić, Fred Douglis, Yves Paindaveine, Richard Wheeler y Songnian Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000.

[MvRT+90] S. J. Mullender, C. van Rossum, A. S. Tanenbaum, R. van Renesse y H. van Stavern. *Amoeba: a distributed operating system for the 1990s.* IEEE Computer, Mayo 1990.

[Ous82] John K. Ousterhout. Scheduling Techniques for Concurrent Systems. En *Third International Conference on Distributed Computing Systems*, páginas 22–30, Mayo 1982.

[SGK+85] Russel Sandberg, David Goldberg, Steve Kleiman, Dan Walsh y Bob Lyon. The Design and Implementation of the Sun Network File System. En *Proceedings of the USENIX Summer Conference*, páginas 119–130, Berkeley, CA, USA, Junio 1985.

[SH96] Peter Smith y Norman C. Hutchinson. Heterogeneous Process Migration: The Tui System. Reporte Técnico TR-96-04, Department of Computer Science, University of British Columbia, Febrero 1996.

[Shi97] Ken Shirriff. Building Distributed Process Management on an Object-Oriented Framework. En *Proceedings of the USENIX Annual Technical Conference (USENIX-97)*, páginas 119–132, Berkeley, Enero 6–10 1997. Usenix Association.

[Tho00] Mary Thompson. Security implications of typical grid computing scenarios, Diciembre 08 2000.